

Aplicaciones Gráficas Utilizando Qt

Materia: Sistemas Operativos

Año: 2011

Web: <http://ccfi.com.ar/moodle>

¿Qué es Qt?

- Qt es un toolkit para generar aplicaciones gráficas usando C++ en varias plataformas (Windows, Linux, MacOS X, Symbian, etc).
- El entorno de desarrollo Qt se divide en varios módulos, además de algunas herramientas de consola:
 - **Qt Designer** es el diseñador de ventanas.
 - **Qt Assistant** es la documentación en línea.
 - **Qt Linguist** es la herramienta de traducción.
 - **QtCreator** IDE para facilitar el desarrollo (*mostrar*)

Módulos de Qt

- **QtCore**

QtCore provee el objeto básico para todos los elementos de Qt: *QObject* permite conectar elementos gráficos (o no) mediante un mecanismo de señales (**signals**) y zócalos (**slots**).

- **QtGui**

Provee casi todos los elementos gráficos y la aplicación:

- QWidget
- QApplication

Aplicación mínima en Qt

La variable `app` es del tipo `QApplication` y se crea (internamente se llama a **new**). La variable `w` es puntero a `QWidget`.

```
#include <QtGui/QApplication>
#include <QtGui/QWidget>

int main(int argc, char **argv){
    QApplication app(argc, argv); // Objeto estático
    QWidget *w = new QWidget(); // Objeto dinámico
    w->setWindowTitle("Mi primer ventana");
    w->show(); // Mostrar ventana
    return app.exec();
}
```

Aplicaciones Gráficas con Qt

Generalmente es más sencillo incluir `<QtCore>` y `<QtGui>`.

Los Widgets son clases de C++

```
// miwidget.h
#include <QtGui> // Agregamos todo QtGui

class MiVentana : public QWidget {
    Q_OBJECT      // Macroprocesador para la magia

public:
    // No asustarse!
    explicit MiVentana(parent = 0): QWidget(parent);
private:
    QPushButton *miBoton;

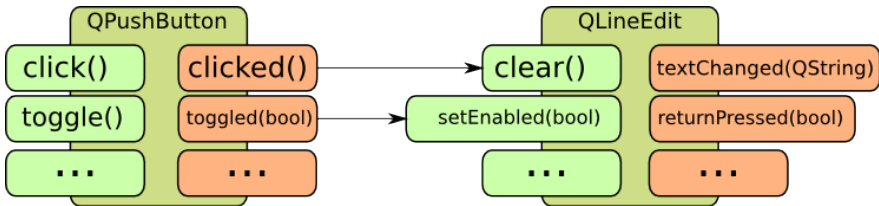
}
```

Por lo general se divide la interfase **miwidget.h** de la implementación **miwidget.cpp**, en la implementación se utiliza el operador **::** para indicar que estamos implementando un método de una clase.

```
/** Constructor, cuando se crea un clase se llama  
 * a este método que tiene el mismo nombre que  
 * la clase, sirve para inicializar.  
 */  
MiVentana::MiVentana(parent = 0): QWidget(parent) {  
    this->miBoton = new QPushButton("Mi botón");  
}
```

Señales y Slots

Una señal es un evento que puede emitir una instancia de un `QObject`, este evento puede ser conectado a uno o más zócalos (**slots**). Una señal puede transportar información y en general las señales son *thread-safe*.



(ver ejemplo de conexión en QtCreator)

Conexión de las señales

Los objetos que heredan de **QObject** permiten conectar eventos mediante un método **QObject::connect**.

```
QObject::connect(boton, SIGNAL(clicked()),  
                 otroObjeto, SLOT(acutalizarInfo()));
```

¿Como depurar?

- `qDebug()` imprime cadenas, acepta `const char *`:

```
// Una cadena simple  
QDebug( "Pase por acá" );  
  
// Un mensaje formateado  
char buffer[255];  
int i = 4;  
snprintf(buffer, 254, "El valor es %d", i);  
QDebug(buffer);
```

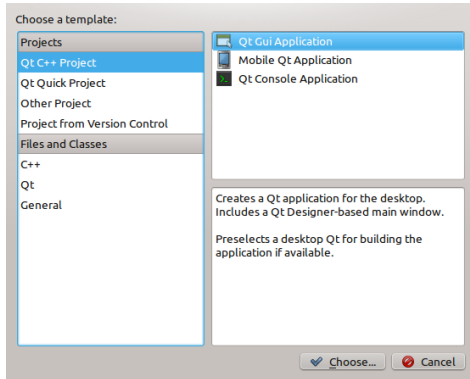
¿Como depurar (2)?

- También se puede usar el operador de flujo << con la función `qDebug()`.

```
QDebug( ) << "El numero es" << QString::number(i);
```

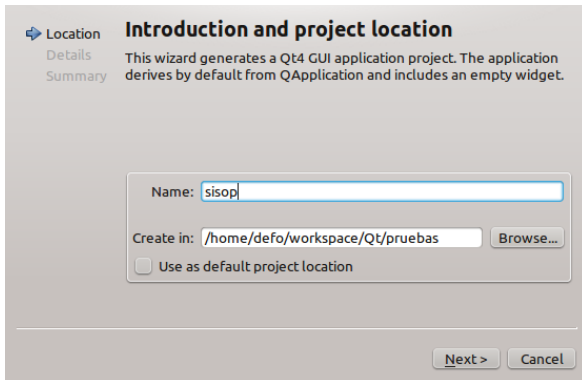
Cómo desarrollar una aplicación

En **QtCreator**, File->New...



Aplicaciones Gráficas con Qt

Definimos la ruta y nombre del proyecto



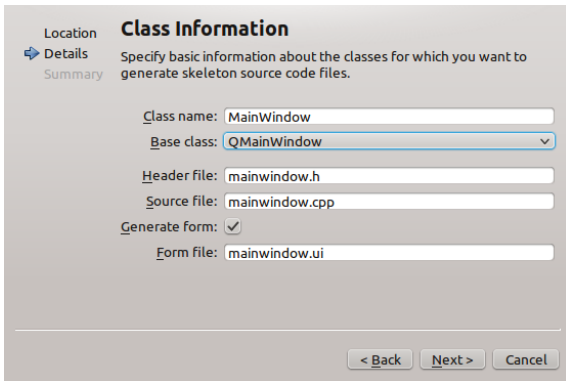
The image shows a Qt4 wizard window titled "Introduction and project location". On the left, there are three tabs: "Location" (selected with a blue arrow), "Details", and "Summary". The main text area states: "This wizard generates a Qt4 GUI application project. The application derives by default from QApplication and includes an empty widget." Below this, there is a form with the following fields:

- Name:** A text input field containing "sisop".
- Create in:** A text input field containing "/home/defo/workspace/Qt/pruebas". To the right of this field is a "Browse..." button.
- ☐ Use as default project location

At the bottom right of the window, there are two buttons: "Next >" and "Cancel".

Aplicaciones Gráficas con Qt

Definimos la clase principal



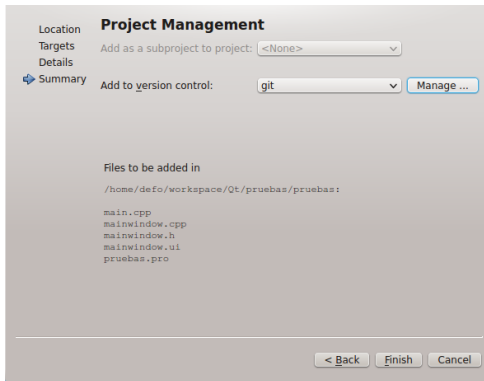
The image shows the 'Class Information' dialog box in Qt Creator. On the left, there are three tabs: 'Location', 'Details' (which is selected and highlighted with a blue arrow), and 'Summary'. The main area is titled 'Class Information' and contains the following fields:

- Class name:** A text field containing 'MainWindow'.
- Base class:** A dropdown menu with 'QMainWindow' selected.
- Header file:** A text field containing 'mainwindow.h'.
- Source file:** A text field containing 'mainwindow.cpp'.
- Generate form:** A checkbox that is checked.
- Form file:** A text field containing 'mainwindow.ui'.

At the bottom right, there are three buttons: '< Back', 'Next >', and 'Cancel'.

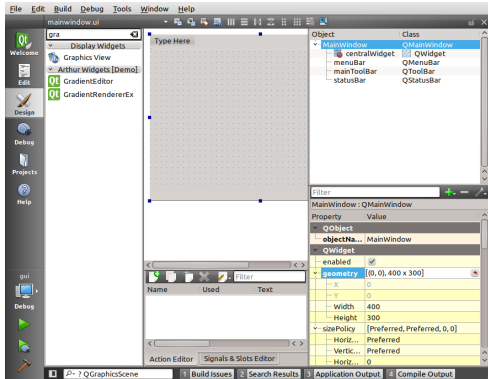
Aplicaciones Gráficas con Qt

Aceptemos el resumen de la aplicación y definimos el sistema de control de versiones si está disponible (no es necesario).



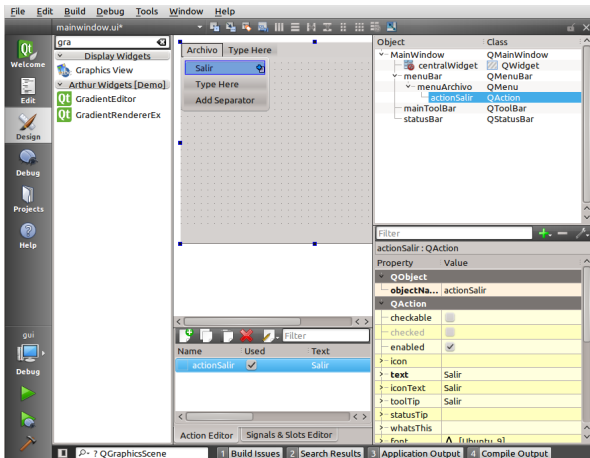
Diseñado de la GUI

Diseño de la GUI



Aplicaciones Gráficas con Qt

Definición del menú

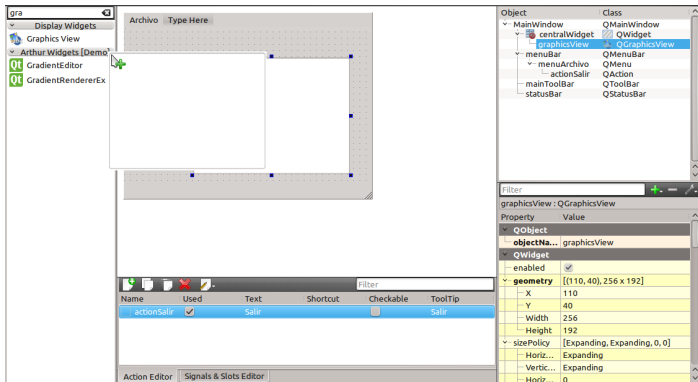


Finalmente para conectar la acción

```
MainWindow::MainWindow(QWidget *parent) :  
    QMainWindow(parent),  
    ui(new Ui::MainWindow)  
{  
    ui->setupUi(this);  
    // Conexión  
    QObject::connect(this->ui->actionSalir,  
        SIGNAL(triggered()), qApp, SLOT(quit()));  
}
```

Utilizando un QGraphicsView

Para agregar un QGraphicsView lo arrastramos desde la paleta de widgets de la izquierda.



Definiendo una escena

Ahora que tenemos un visor(QGraphicsView), necesitamos una escena, para esto la agregamos como atributo a la clase (en mainwindow.h):

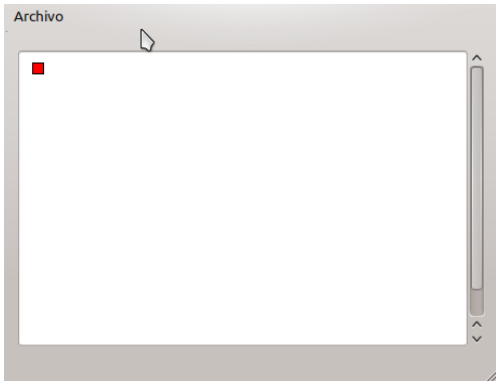
```
#include <QtGui>
class MainWindow : public QMainWindow
{
    Q_OBJECT
    // ...
private:
    Ui::MainWindow *ui;
    QGraphicsScene *escena;
};
```

Instanciando la escena

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    // Conexion
    QObject::connect(this->ui->actionSalir, SIGNAL(triggered()), qApp, SLOT(quit()));
    this->escena = new QGraphicsScene(0, 0, 400, 300);
    this->ui->graphicsView->setScene(escena);
    // Agregar una rectangulo :)
    this->escena->addRect(10, 10, 10, 10, QPen("black"), QBrush("red"));
}
```

El resultado

Ahora la aplicación cuenta con un ítem, en este caso un rectángulo, pero se pueden utilizar imágenes.



Utilizando imágenes en los ítem

Para usar imágenes, hay que crear un archivo de recursos, desde:

- En **File->New**, elegir en `Files and Classes`, **Qt Resource file**
- Elegir un nombre y a continuación en la ventana del recurso
 - Agregar un prefijo, ej: imágenes (*prefix*)
 - Agregar una o más imágenes ej: caballo.png
- Llamar a `addPixmap`

```
QGraphicsItem i = escena->addPixmap(QPixmap(":/imagenes/caballo.png"))
```

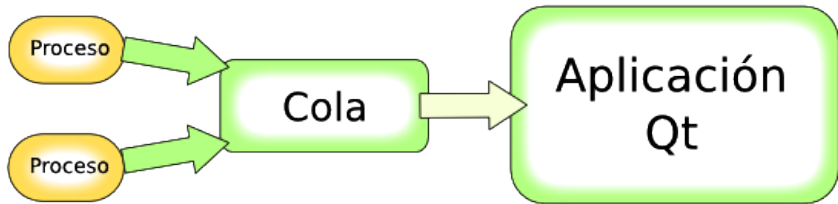
Comunicación de Qt y Colas

Una aplicación en C++, por ende cualquier aplicación Qt, puede hacer uso de las funciones de C, en particular nos enfocamos en las funciones `mq_open`, `mq_send`, `mq_receive`.

Para comunicar con un proceso que envía mensajes, definimos un tipo `Mensaje` común a las dos aplicaciones (C y Qt/C++), lo haremos en `mq/comun.h`.

Esquema de comunicación

Para generar una simulación, los procesos utilizaran colas de mensajes para notificar a la interfase de sus cambios de estado.



Un tipo de mensaje común

Para la aplicación de ejemplo, vamos a utilizar un mensaje común a la aplicación Qt y la apelación en C.

Lo definimos de la siguiente manera:

```
typedef struct {  
    int quien; // Para el pid  
    int tipo;  // Tipo de mensaje, identifica  
               // de que evento se trata  
} Mensaje;
```

Identificación de eventos

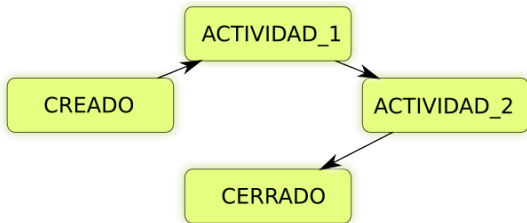
Cada mensaje enviado a la cola se identificará con el pid y transmitirá algún tipo de evento.

```
#define TIPO_CREADO          0
#define TIPO_ACTIVIDAD_1    1
#define TIPO_ACTIVIDAD_2    2
#define TIPO_CERRADO        3
```

Estas constantes representan que el proceso informa cambios de estado:

Diagrama de estados del programa

El siguiente diagrama ejemplifica la secuencia en la que cada proceso simulador enviará los mensajes, intercalando una espera aleatoria en las transiciones.



Código del envío

Código del cliente

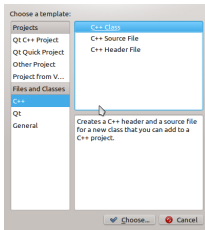
```
Mensaje m;  
m.quien = getpid();  
m.tipo = TIPO_CREADO;  
mq_send(cola, (const char *)&m, sizeof(m), 0);  
esperaAleatoria();  
  
m.tipo = TIPO_ACTIVIDAD_1;  
mq_send(cola, (const char *)&m, sizeof(m), 0);  
esperaAleatoria();  
  
m.quien = getpid();
```

```
m.tipo = TIPO_ACTIVIDAD_2;  
mq_send(cola, (const char *)&m, sizeof(m), 0);  
esperaAleatoria();  
  
m.quien = getpid();  
m.tipo = TIPO_CERRADO;  
mq_send(cola, (const char *)&m, sizeof(m), 0);  
esperaAleatoria();
```

Recibir mensajes

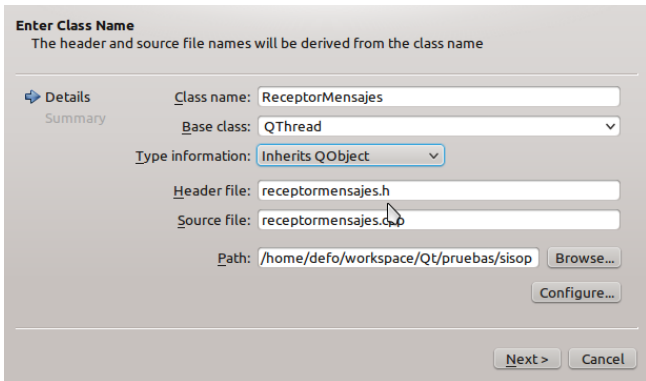
Para recibir mensajes no podemos quedarnos bloqueados en un `mq_receive()`, por lo que necesitamos un hilo. Para crear un hilo, necesitamos extender de `QThread` e implementar el método `run()`

Para crear una clase que herede de `QThread` desde File->New, elegimos **C++ Class**.



Aplicaciones Gráficas con Qt

Definición de la clase, en este caso la bautizamos ReceptorMensajes



The image shows the 'Enter Class Name' dialog box in Qt Creator. The title bar is 'Enter Class Name'. Below the title, it says 'The header and source file names will be derived from the class name'. There are two tabs: 'Details' (selected) and 'Summary'. Under the 'Details' tab, there are several fields: 'Class name' with the text 'ReceptorMensajes', 'Base class' with a dropdown menu showing 'QThread', 'Type information' with a dropdown menu showing 'Inherits QObject', 'Header file' with the text 'receptormensajes.h', 'Source file' with the text 'receptormensajes.cpp', and 'Path' with the text '/home/defo/workspace/Qt/pruebas/sisop'. There are 'Browse...' and 'Configure...' buttons next to the 'Path' field. At the bottom right, there are 'Next >' and 'Cancel' buttons.

Enter Class Name
The header and source file names will be derived from the class name

Details Summary

Class name: ReceptorMensajes

Base class: QThread

Type information: Inherits QObject

Header file: receptormensajes.h

Source file: receptormensajes.cpp

Path: /home/defo/workspace/Qt/pruebas/sisop

Browse... Configure...

Next > Cancel

Ejemplo

Ver ejemplo en la carpeta `sisop`:

- Dentro de la carpeta `mq` se encuentra *simulador.c* encargado de enviar mensajes.
- En `mq/comun.h` está definida la estructura de `Mensaje`.
- Los mensajes son recibidos en el método `run` de la clase `ReceptorMensajes` (`receptormensajes.h/receptormensajes.cpp`).
- Cada vez que llega un mensaje, se emite una señal `recepcionMensaje(int quien, int tipo)`

Ejemplo (2)

- La señal `recepcionMensaje(int, int)` está conectado con un slot con el mismo nombre y está implementada en la ventana principal.
- Cuando el tipo de mensaje es (ver slot `recepcionMensaje` de `mainwindow.cpp`):
 - `TIPO_CREADO` Crear un `QGraphicsRectItem*` con `escena->addRect`
 - `TIPO_ACTIVIDAD_1` y `TIPO_ACTIVIDAD_2` Cambia el color del `QGraphicsRectItem*` correspondiente
 - `TIPO_CERRADO` Elimina el ítem de la escena

Ejemplo (3)

- La ventana principal tiene un arreglo, diccionario, hashmap (asociación clave-valor)

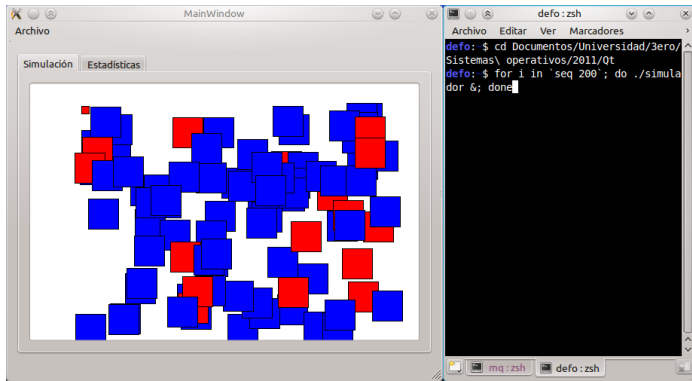
```
QMap<int, QGraphicsRectItem*>  
asociacionPidItem;
```

En esta estructura de datos se asocia el pid con el item (ver de cambiar el tipo para utilizara `QGraphicsPixmapItem`).

Ver `mainwindow.h`.

- Como ejemplo de manejo de estadísticas, en una de las solapas se actualiza una etiqueta (`QLabel`), con la cantidad de procesos que recibe la interfase gráfica.

Ejemplo (4)



Ejemplo (5)

Recordar aumentar el tamaño máximo de el mensaje de la cola y la cantidad máxima de mensajes:

```
echo 512 | sudo tee /proc/sys/fs/mqueue/msg_max
```

```
echo 512 | sudo tee /proc/sys/fs/mqueue/msgsize_max
```